LDAS Driver Design

Introduction

The modification of the current LDAS driver is intended to make use of the advanced features of fortran 90 programming language, which are especially useful for object oriented programming. The latest standard, Fortran 2000 is expected to provide full object oriented features and porting over to Fortan 2000 will be easier if the current code is designed in the object oriented fashion. Further, the modular structure of the code and the use of object oriented features would facilitate easy migration over to another object oriented language such as C++.

Fotran 90 provides a number of features which appear to be useful for OOP. Almost all the features of an object oriented language such as C++ can be emulated in fortran with the exception of dynamic binding. However Decyk et al. [1] presents techniques to simulate dynamic binding in Fortran 90. These techniques are employed in this modification of the ldas driver.

Modified LDAS Driver

A snippet of the "old" ldas main driver is shown in Figure 1. It can be seen that there are lot of "if" statements for each conditions, be it a land surface model related or boundary conditions related. The modified driver 2 eliminates these conditionals, and delegates the actual work to the respective modules (see Figure 3). A future modification of the code such as inclusion of a new land surface model, or a new forcing will not necessitate a change in the main driver. The "hook points" for the incorporation of new procedures are well defined through different interfaces. For example, the incorporation of a new land surface model such as VIC would involve the modification of the subroutines in the lsm_module.

The current design uses many object oriented concepts such as derived data types, data encapsulation, function overloading, dynamic binding, and classes. The interface statements in Fortran90 allow procedures in different classes (modules) to have same

```
 ......
 ......
 !=== Get LDAS Base Forcing
       if(ldas%feta.eq.1)  call geteta(ldas,grid)
       if((ldas%fncep.eq.1).or.(ldas%fnasa.eq.1))
     &     call getncep(ldas,grid)
       if(ldas%fgdas.eq.1) call getgdas(ldas,grid)
       if(ldas%fgeos.eq.1) call getgeos(ldas,grid)
       if(ldas%fr_ecmwf.eq.1) call getreanlecmwf(ldas,grid)

 ......
 ......
!=== Call CLM Main Subroutine

       if(ldas%rclm.eq.1)then
        do t=1,ldas%nch
          call clm_main(clm(t),drv%day)
        enddo
       endif

!=== Call NOAH LSM Main Subroutine

       if(ldas%rnoah.eq.1)then
        do t=1,ldas%nch
          call noah_main(t,ldas,tile(t),noah(t))
        enddo
       endif
```

Figure 1: The “old” ldas main driver

function names. Since dynamic binding has to be emulated in software, it requires the creation of appropriate polymorphic classes.

References

[1] V. K. Decyk, C. D. Norton, and B. K. Szymanski. How to express c++ concepts in fortran 90. Technical report, Renesselaer Polytechnic Institute, Scientifc Computation Research Center.

```
      .....

      call LIS_lsm_init(ldas_drv)
      call LIS_baseforcing_init(ldas_drv)
      .....
      .....
         call LIS_get_base_forcing(ldas_drv)
....
....

!==== Tile Loop & Run Models
         do t=1, getnch(ldas_drv)
            call LIS_lsm_main(t,ldas_drv)
         enddo
....
....
```

Figure 2: The “modified” ldas main driver

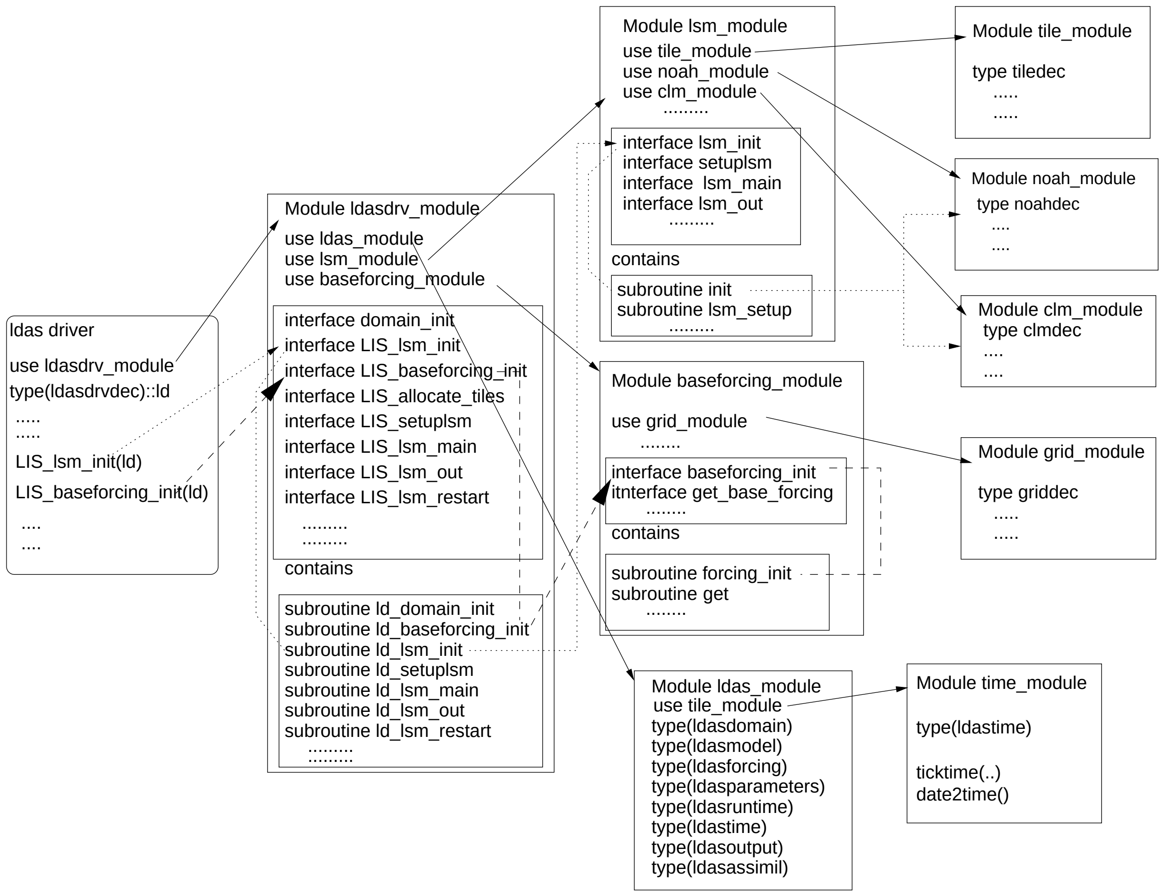

Figure 3: Design of the LDAS driver and the delegation of function calls